CS412 Machine Learning Spring 2024 Homework 3 Report

Kanat Özgen

1. Data Pre-processing
   1. Stratification of the Data

The data was stratified using the stratify = labels statement during test-train split. The data was later split across the test data once more, to create the validation dataset.

* 1. StandardScaler

The StandardScaler object was first fit for the training dataset. After that, the same object is used to transform the test dataset.

1. Hyperparameter Tuning
   1. Implementation
      1. Sigmoid Function

The sigmoid function sets the accuracy result between a boundary, which is zero and one. The implementation of the sigmoid function can be found below:

def sigmoid(z):

    return 1/(1+np.exp(-z))

* + 1. Cost Function

The cost (loss) function finds the actual value of the function created by the weight values. This value itself does not hold a significance. The change rate of this function determines the change of the weight values. The implementation of the cost function is as follows:

def cost\_function(X, y, w):

    m = len(y)

    z = np.dot(X, w)

    h = sigmoid(z)

    cost = (-1/m)\*np.sum(y\*np.log(h) + (1-y)\*np.log(1-h))

    return cost

* + 1. Gradient Descent
       1. First Gradient Descent Function

In order to accommodate for the need of debugging and logging the cost function and to see whether it is converging to a value or not, this function was implemented. It both returns the training loss and validation loss. The implementation of this function is as follows:

def gradient\_descent1(X, y, x\_val, y\_val, w, alpha, num\_iters):

    m = len(y)

    cost\_history1 = np.zeros(num\_iters)

    cost\_history2 = np.zeros(num\_iters)

    for i in range(num\_iters):

        z = np.dot(X, w)

        h = sigmoid(z)

        w = w - (alpha/m)\*np.dot(X.T, h-y)

        cost\_history1[i] = cost\_function(X, y, w)

        cost\_history2[i] = cost\_function(x\_val, y\_val, w)

return w, cost\_history1, cost\_history2

* + - 1. Second Gradient Descent Function

This function is used for the development of the final model. Implementation is as follows:

def gradient\_descent2(X, y, w, alpha, num\_iters):

    m = len(y)

    for \_ in range(num\_iters):

        z = np.dot(X, w)

        h = sigmoid(z)

        w = w - (alpha/m)\*np.dot(X.T, h-y)

    return w

* 1. Tuning Results

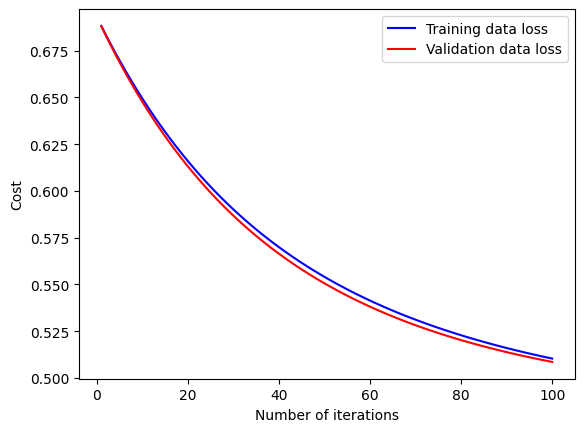
The tuning results show that a minima point is reached whether the step size is correctly determined or not. The minima point decreases the gradient to zero, which stops the update of the weights.

Figure 1: Step size is set to 0.1 and 100 iterations are made.

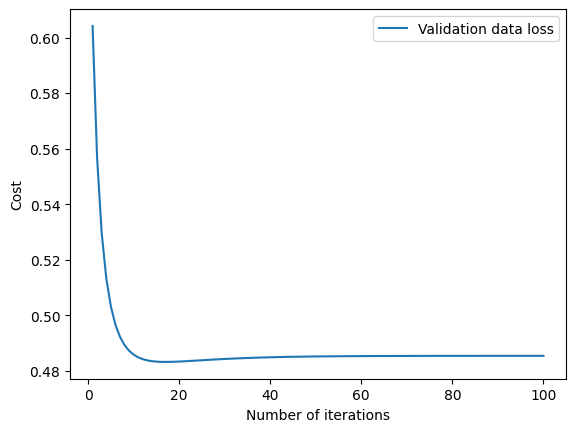
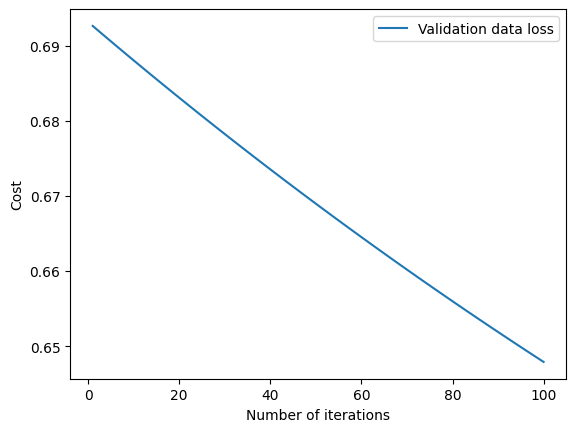
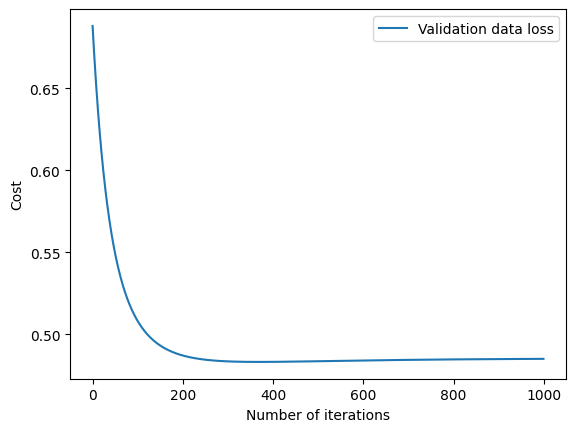


Figure 2: Step size is set to 0.05 and step size to 1000. Model converges after 200 iterations.

Figure 3: Step size is set to 1.

Figure 4: Step size is set to 0.005

As can be seen from the figures, the model converges to a continuous loss of 0.48. This happens because a minima point is already reached and model parameters are not updating anymore.

1. Model Evaluation

A final model using 0.05 as alpha and 200 as iteration number is trained, by combining the validation and training datasets. The model evaluation shows that:

Precision: 0.675

Recall: 0.782608695652174

F1 score: 0.7248322147651006

--------------------

Accuracy: 0.770949720670391